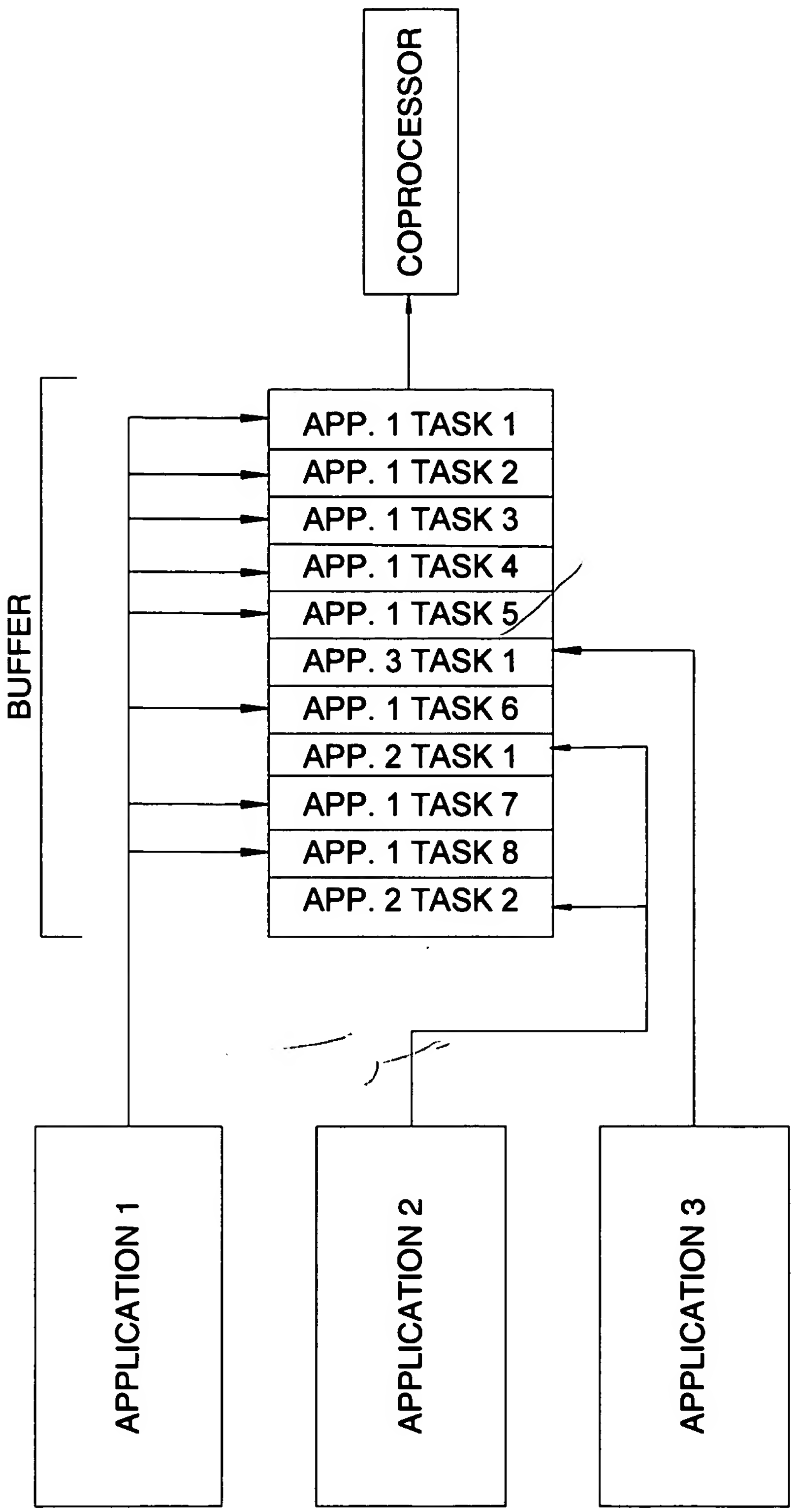


FIGURE 1
(prior art)

Docket No.: MSFT-2857/304862.02 App. No.: Not Yet Assigned Filed: Herewith
Title: MULTITHREADED KERNEL FOR GRAPHICS PROCESSING UNIT
Inventors: Anuj B. G salia and Steve Pronovost
Attorney: Nathaniel Ari Long Phone: (215) 568-3100

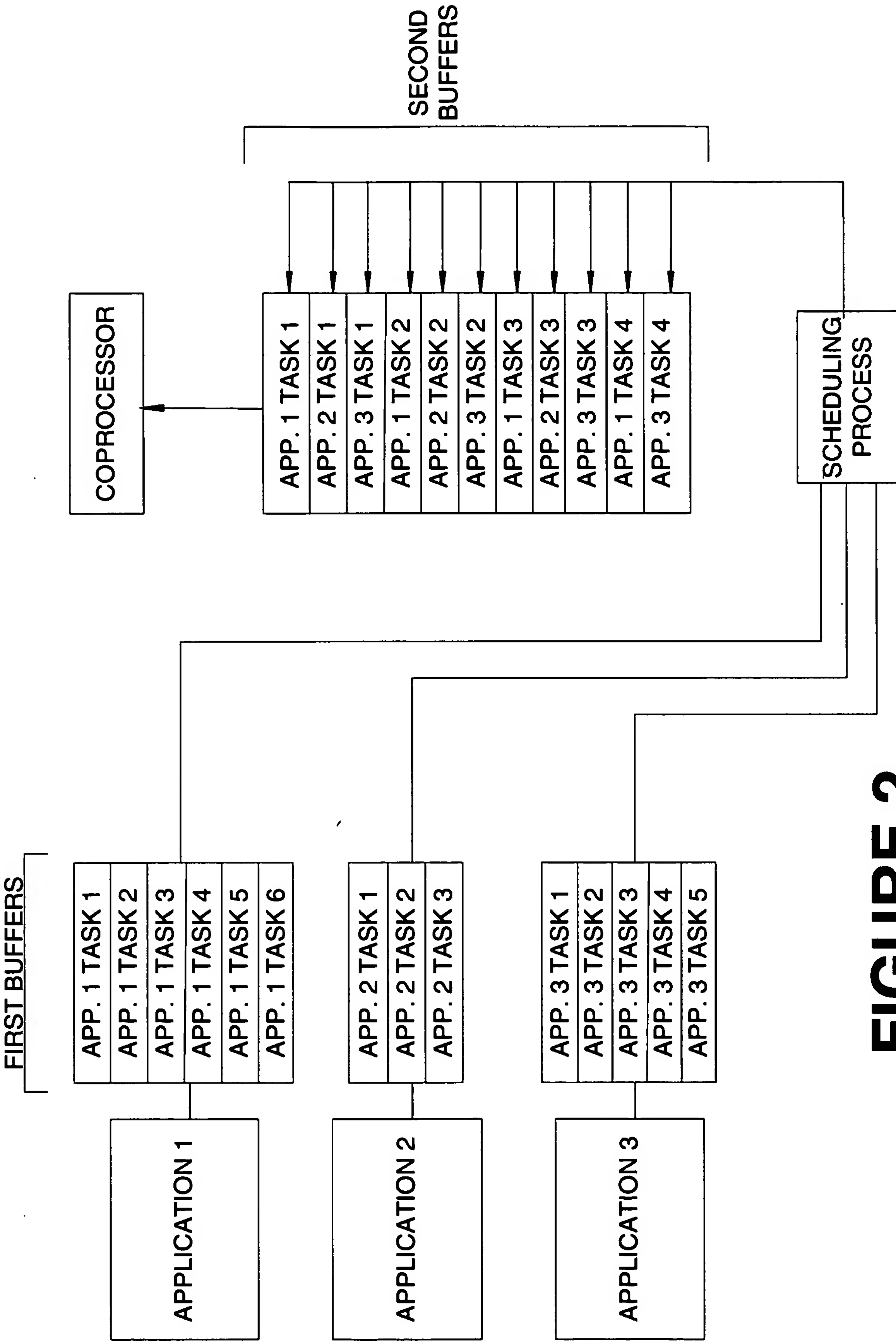


FIGURE 2

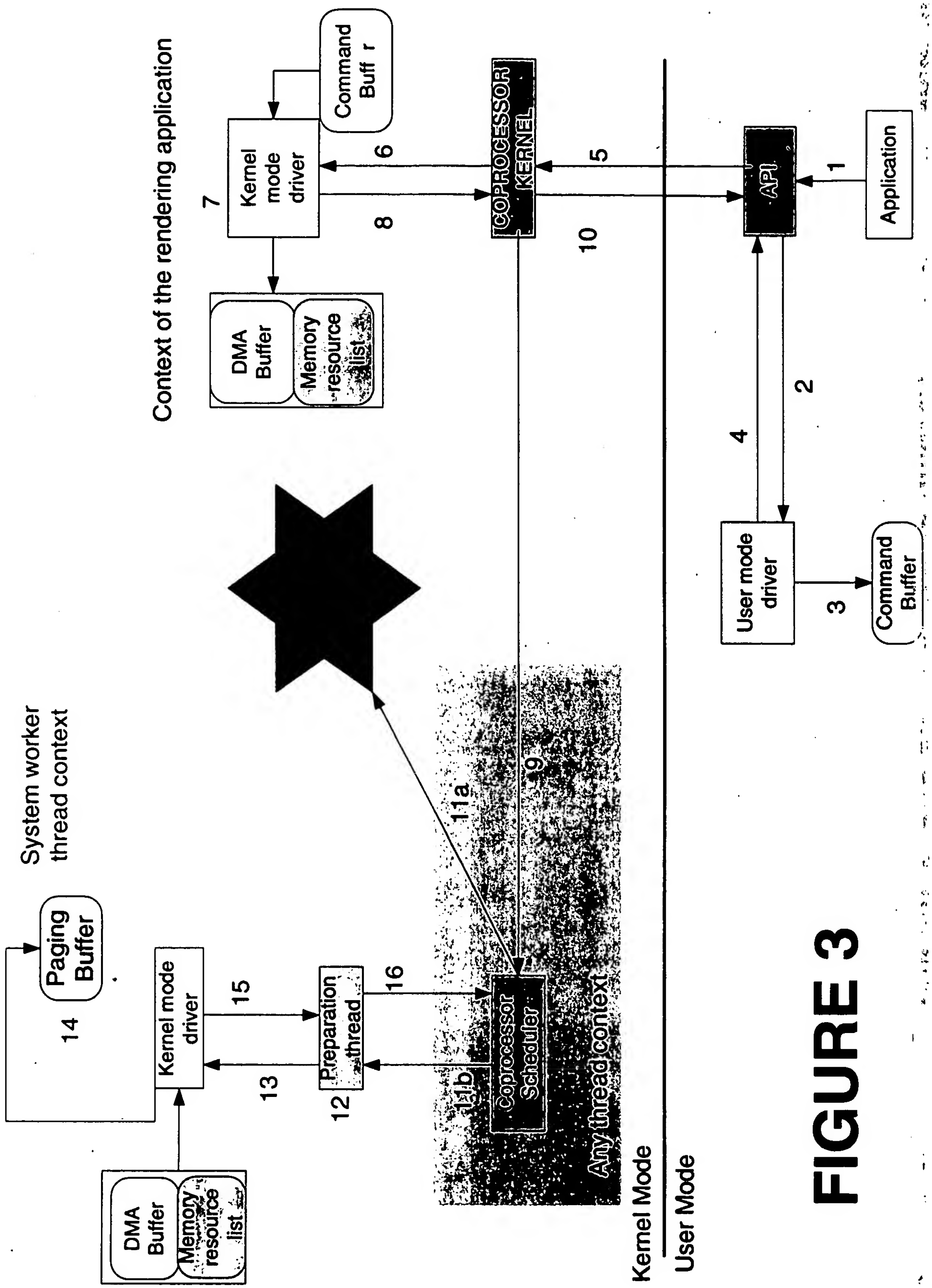

Context of the rendering application
Command Buff r
Kernel mode driver
7
6
8
DMA Buffer
Memory resource list
COPROCESSOR KERNEL
5
10
API
1
Application
2
4
User mode driver
3
Command Buffer
System worker thread context
Paging Buffer
14
Kernel mode driver
15
13
Preparation thread
12
16
11a
9
11b
Coprocessor Scheduler
Any thread context
DMA Buffer
Memory resource list
Kernel Mode
User Mode

FIGURE 3

Exemplary algorithm

PROCESS A: Submit (irql passive, rendering thread context)

Allocate DMA buffer.

- The DMA buffer may be allocated in a system memory aperture.

Copy and validate user mode buffer into DMA buffer.

Build list of any required or desired memory resources and list of potential split point(s) for the DMA buffer if not all memory resources can be paged in at the same time.

If the ready list is empty and no DMA buffer is currently being prepared:

- If all the memory resources for the current DMA buffer are already present in memory, ask the driver to patch the DMA buffer and submit it to the hardware;
- If some memory resources need to be paged-in, submit the DMA buffer to the worker thread.

PROCESS B: Quantum expires (irql device, any thread context)

If the current task is still processing its paging buffer:

- Allow the current task to continue running, set the current quantum as expired.

If next task is ready to be run:

- Start next task;
- Reset the quantum to "running" (not expired);
  - If the task requires importing memory resources, submit it to the worker thread;
  - If all memory resources are present, ask the driver to patch the next DMA buffer.

If next task is not ready to be run:

Allow the current task to continue running, set the current quantum to "expired."

FIGURE 4(A)

Exemplary algorithm

PROCESS C: Task finishes (irql device., any thread context)

If the next task is ready:

Start next task;

Reset the quantum to “running”;

Choose the next task to run.

If the task requires importing memory resources, submit it to the worker thread.

If all memory resources are present, ask the driver to patch the next DMA buffer.

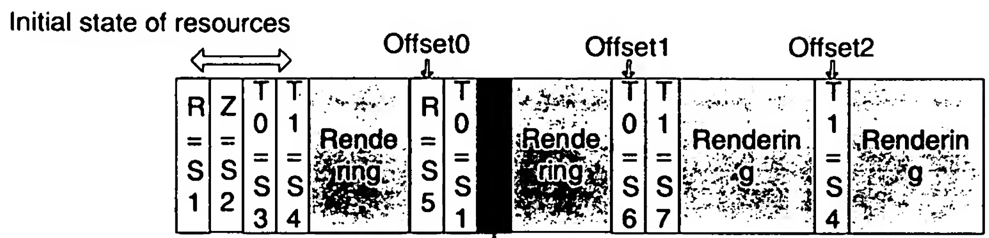
If the next task is not ready:

Stall the coprocessor while the worker thread completes the next task setup or until an application submits a new buffer;

During worker thread completion, the priority of the worker thread is booted so the worker thread finishes its work as soon as possible.

FIGURE 4(B)

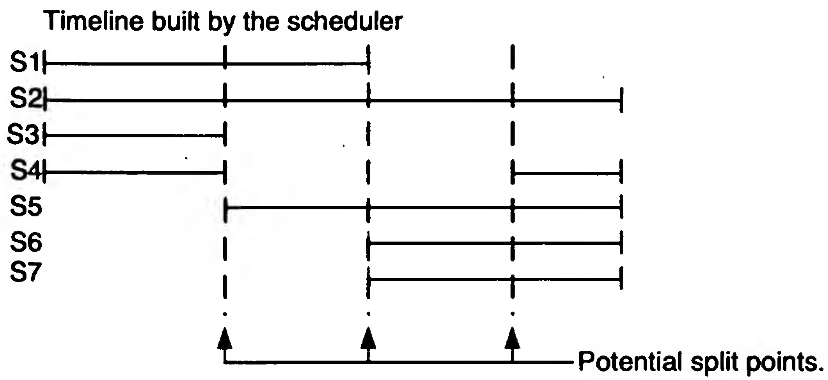
DMA Buffer



Pseudo command (not necessarily in the DMA buffer as a memory resources programmation). This is a marker used by the driver to indicate that this memory resource isn't used beyond this point (for example, a texture stage state where it was bounded may have been deactivated).

Memory resource list

Handle	ResourceId	Offset
S1	0	0
S2	1	0
S3	2	0
S4	3	0
S5	0	Offset0
S1	2	Offset0
S6	2	Offset1
S7	3	Offset1
S4	3	Offset2



Legend
R = Render target
Z = Z-buffer
T0 = Texture in stage 0
T1 = Texture in stage 1
S# = Memory resources #

FIGURE 5

Exemplary algorithm

Provide a list of memory resources to a supplemental memory manager;

The supplemental memory manager returns a list of memory resources that be moved, optionally specifying a move location. This list is called a "command list" for purposes of this description.

If the video memory manager can't find a location for memory resources in the list, it can chose some allocated memory to evict;

The supplemental memory manager can go through the list and inspect the segments;

If the current process working set is below the commit limit in that segment.

Mark some surfaces for eviction according to the current eviction policy;

Increase the current eviction policy to be harsher next time around;

Go back to the first step and try allocating suplemental memory again;

If the current process is requiring more memory than the current commit limit;

Split the DMA buffer based on the split point built from the timetable for memory resources;

If the DMA buffer can't be split further but not all the memory resources have been marked for eviction yet;

Mark everything for eviction;

Reduce the priority of the application causing the trashing so it doesn't cause as much trouble in the future;

Go back to the first step and try allocating supplemental memory again;

If the DMA buffer can't be split further and all memory resources have been marked for eviction already;

Reject the current DMA buffer for the application. Since this DMA buffer may have contained important state metrics, this process can be discontinued, and the application can be informed of the error.

FIGURE 6

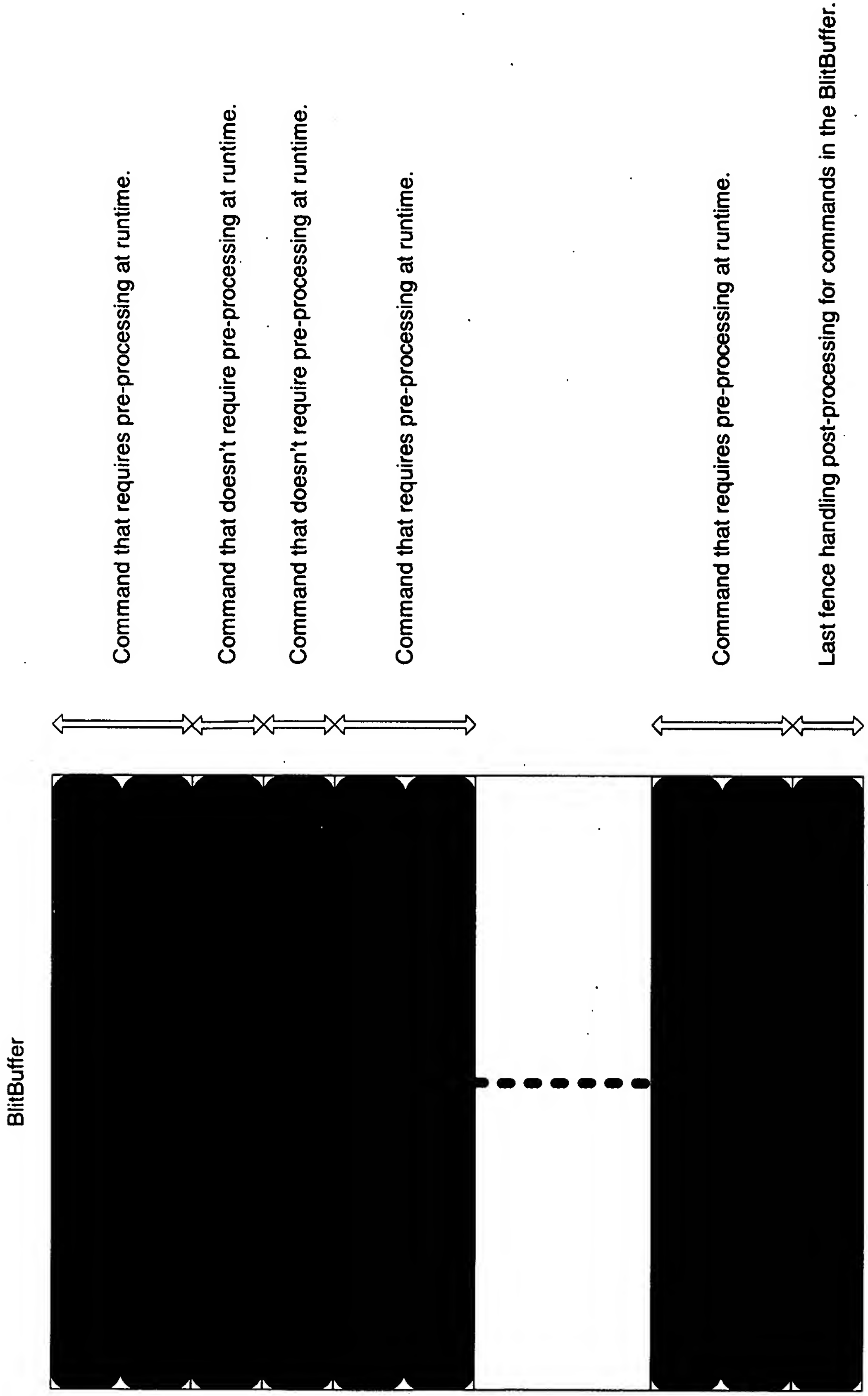

BlitBuffer
Command that requires pre-processing at runtime.
Command that doesn't require pre-processing at runtime.
Command that doesn't require pre-processing at runtime.
Command that requires pre-processing at runtime.
Command that requires pre-processing at runtime.
Last fence handling post-processing for commands in the BlitBuffer.

FIGURE 7

Exemplary algorithm

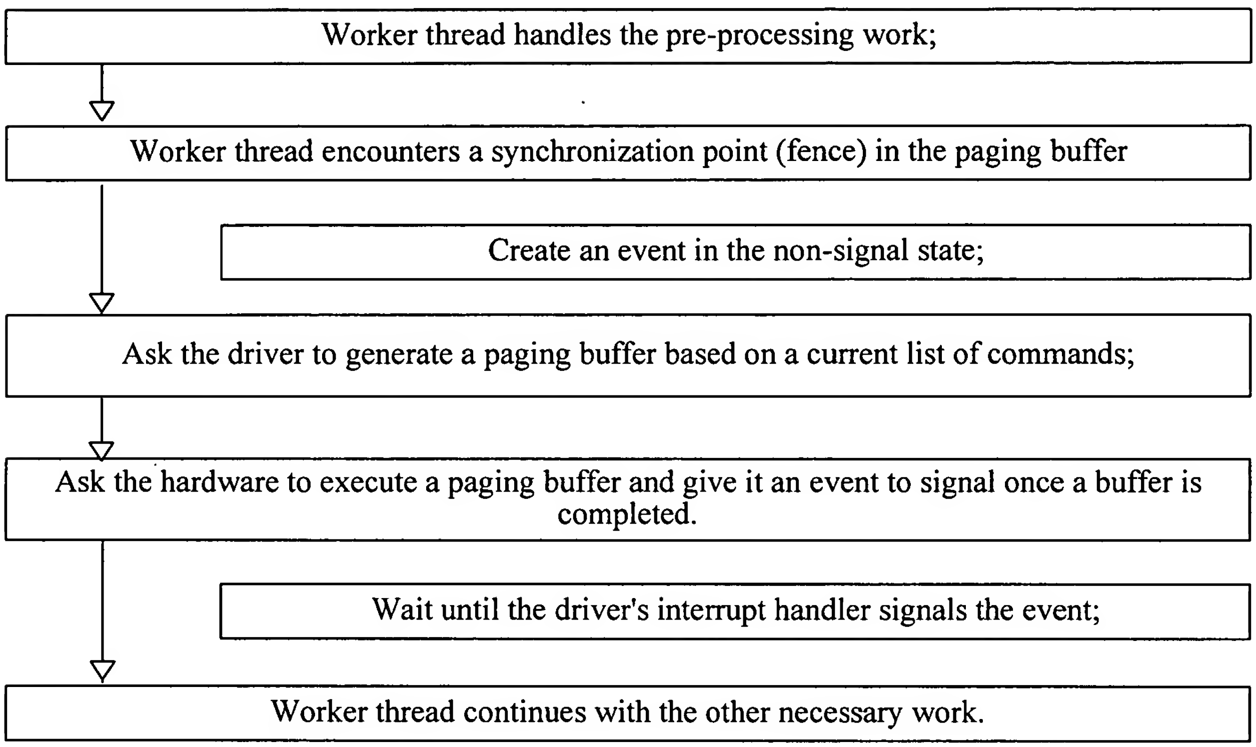


FIGURE 8

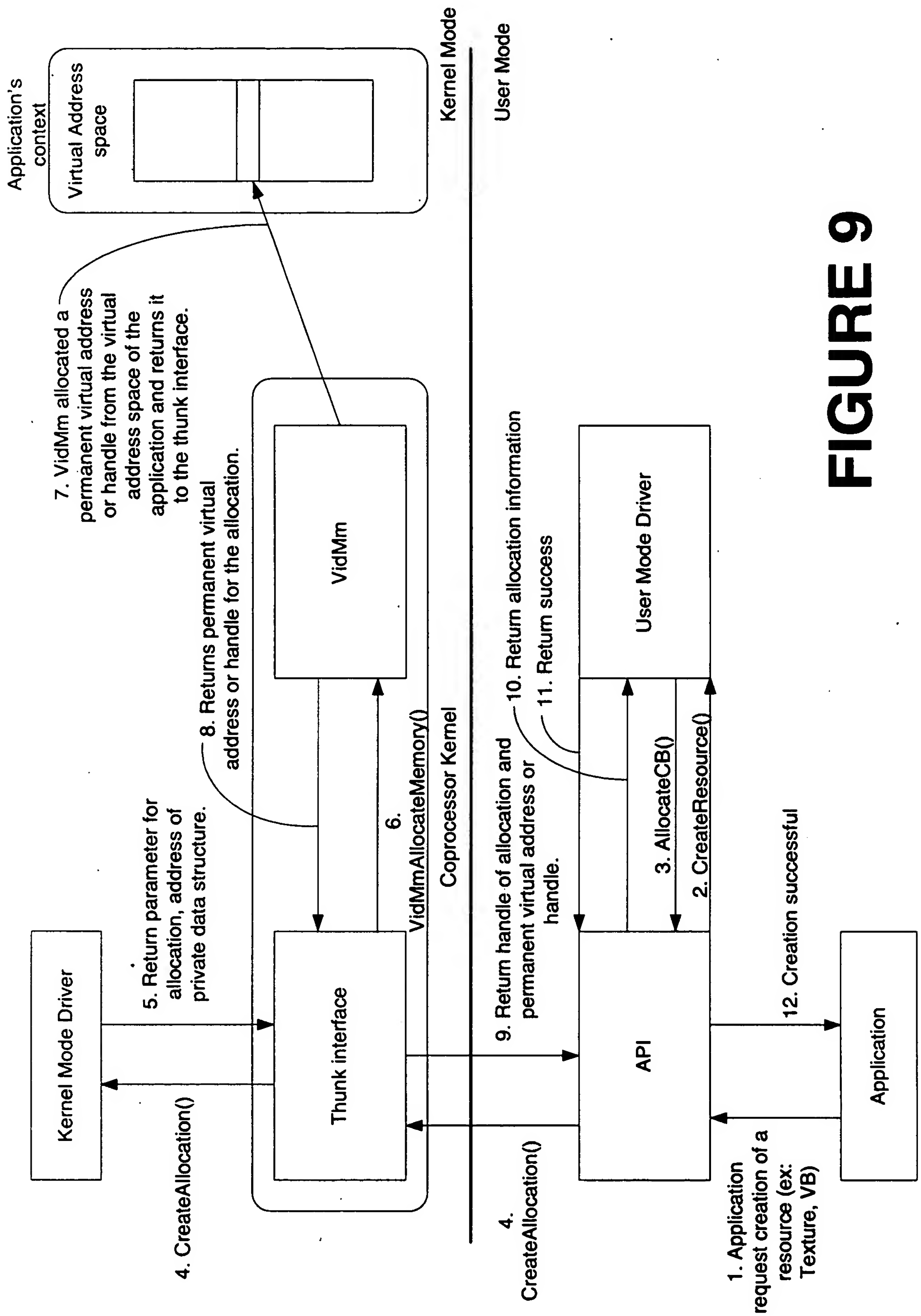

Application's context
Virtual Address space
Kernel Mode
User Mode
7. VidMm allocated a permanent virtual address or handle from the virtual address space of the application and returns it to the thunk interface.
VidMm
8. Returns permanent virtual address or handle for the allocation.
10. Return allocation information
11. Return success
User Mode Driver
Kernel Mode Driver
5. Return parameter for allocation, address of private data structure.
6. VidMmAllocateMemory()
Coprocessor Kernel
9. Return handle of allocation and permanent virtual address or handle.
3. AllocateCB()
2. CreateResource()
12. Creation successful
Thunk interface
API
Application
4. CreateAllocation()
4. CreateAllocation()
1. Application request creation of a resource (ex: Texture, VB)

FIGURE 9

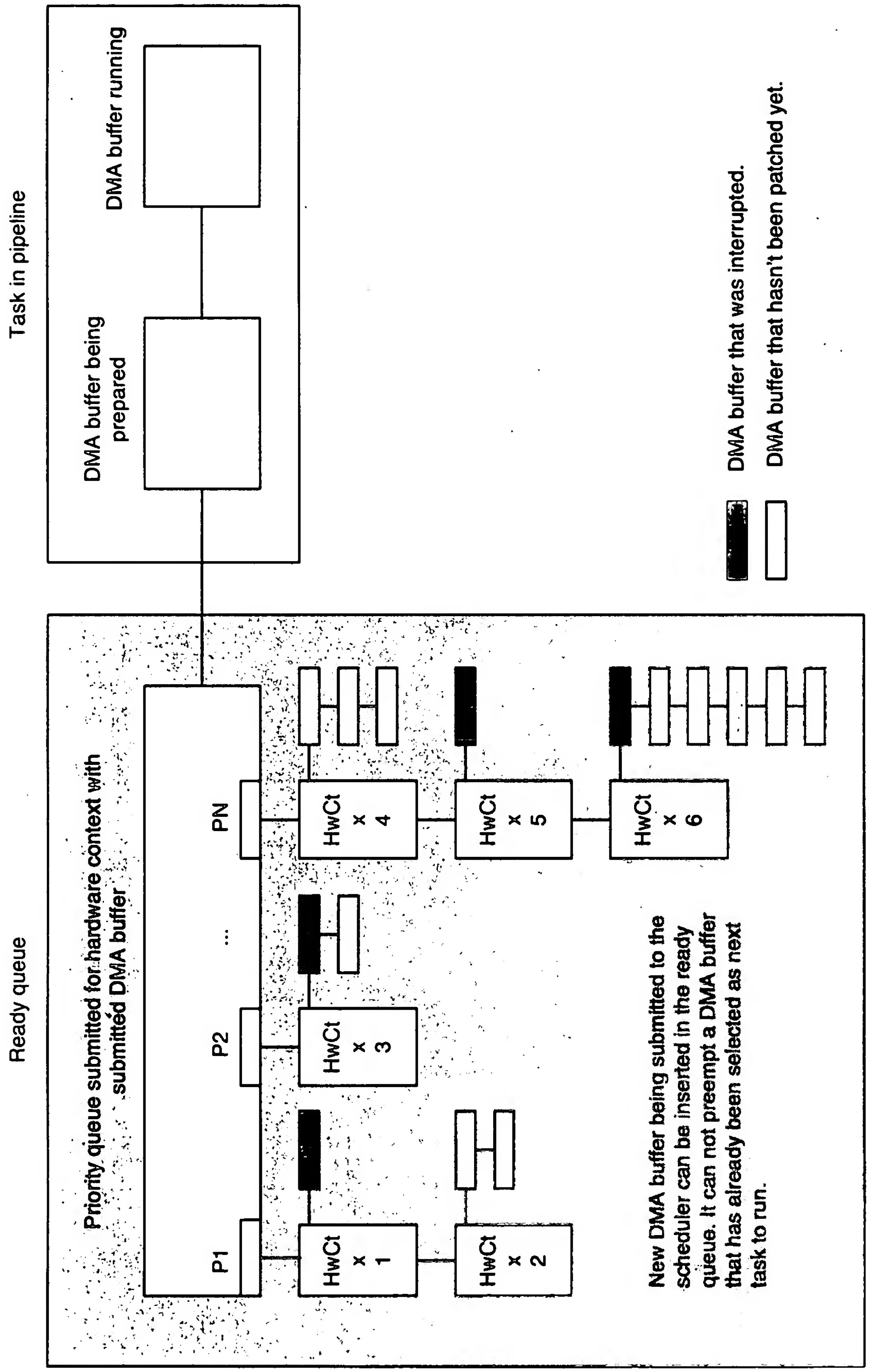

Task in pipeline
DMA buffer being prepared
DMA buffer running
Ready queue
Priority queue submitted for hardware context with submitted DMA buffer
P1
P2
...
PN
HwCt x 1
HwCt x 2
HwCt x 3
HwCt x 4
HwCt x 5
HwCt x 6
New DMA buffer being submitted to the scheduler can be inserted in the ready queue. It can not preempt a DMA buffer that has already been selected as next task to run.
DMA buffer that was interrupted.
DMA buffer that hasn't been patched yet.

FIGURE 10

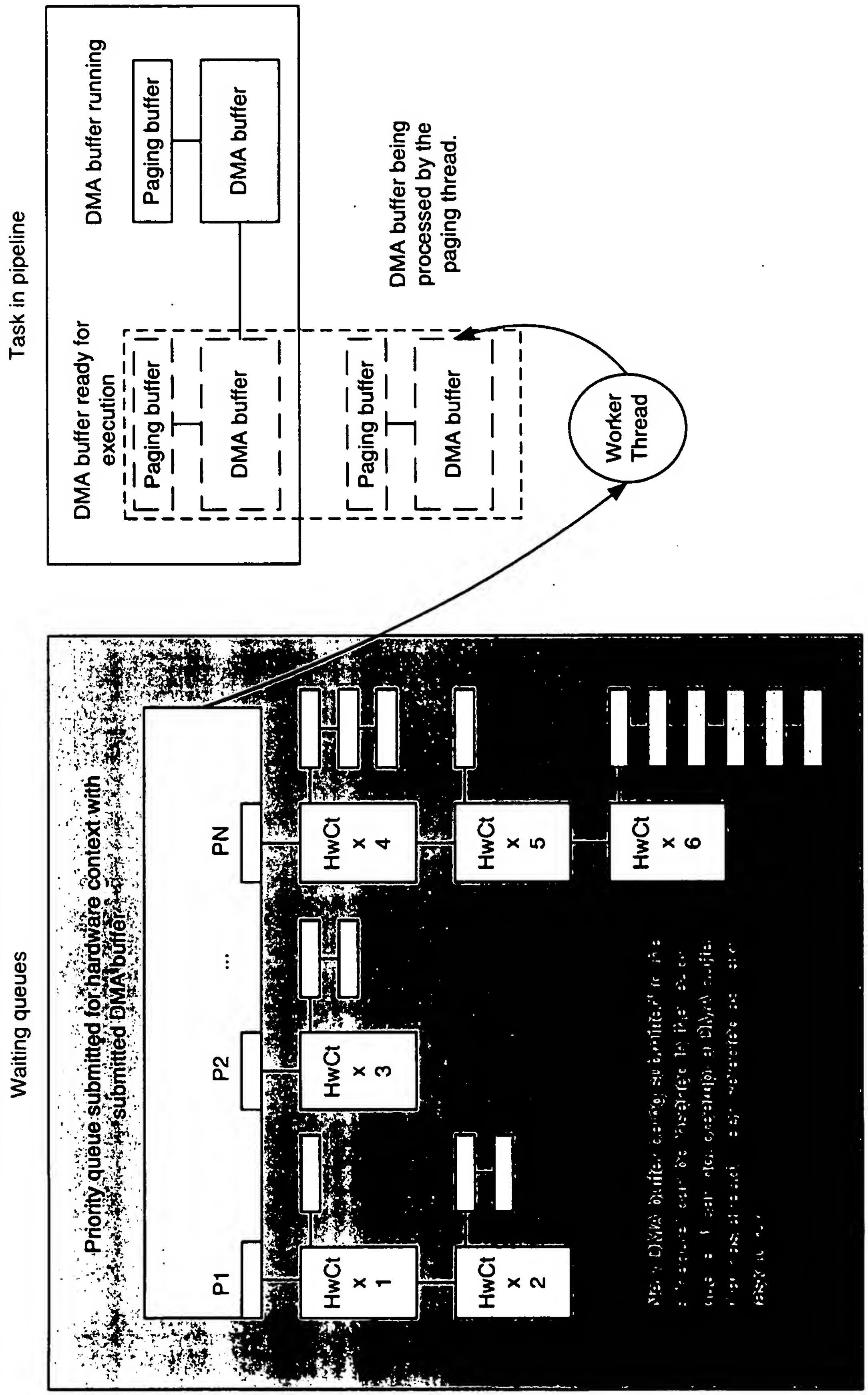

Task in pipeline
DMA buffer running
Paging buffer
DMA buffer
DMA buffer ready for execution
Paging buffer
DMA buffer
Paging buffer
DMA buffer
DMA buffer being processed by the paging thread.
Worker Thread
Waiting queues
Priority queue submitted for hardware context with submitted DMA buffer
P1
P2
...
PN
HwCt x 1
HwCt x 2
HwCt x 3
HwCt x 4
HwCt x 5
HwCt x 6

FIGURE 11

PROCESS A: Submit (IRQL passive, rendering thread context)

If no DMA buffer is being prepared or is ready for execution.
If all the memory resources for the current DMA buffer are already present in memory,
If the coprocessor is idle, give the DMA buffer to the coprocessor.
Else insert the DMA buffer in the ready-to-execute slot.
If some memory resources need to be paged in, submit the DMA buffer to the paging thread.
Else, insert the DMA buffer at the end of the list for the current context.

PROCESS B: Quantum expires (IRQL device, any thread context)

If the current task is still processing its paging buffer,
Allow the current task to continue running.
Set the current quantum as expired.
Else, if next DMA buffer is ready to be run,
Reset the current priority of the current context to its base priority.
Move the current context to the end of the queue for its priority.
Submit next DMA buffer to the coprocessor.
Reset the quantum as being running (not expired).
Choose the next DMA buffer to execute.
If the DMA buffer requires paging, submit it to the paging thread.
Else, all memory resources are already present; just insert the DMA buffer in the ready slot.
Else, the next task isn't ready to be run;
Allow the current task to continue running.
Set the current quantum as expired.

PROCESS C: Task finishes (IRQL device, any thread context)

If next DMA buffer is ready to be run,
Submit next DMA buffer to the coprocessor.
Reset the quantum as being running (not expired).
Choose the next DMA buffer to execute.
If the DMA buffer requires paging, submit it to the paging thread.
Else, all memory resources are already present; just insert the DMA buffer in the ready slot.
Else, the next task isn't ready;
If the paging thread is currently working on the next DMA buffer, boost the priority of the worker thread temporarily so it finishes its work as soon as possible.

FIGURE 12(A)

PROCESS D: Paging thread (IRQL passive, system thread)

```
Set current eviction policy to first policy.
Ask the memory manager to page in the resource list.
If all the resource were paged in successfully,
    Move the paging buffer and DMA buffer to the ready-to-execute slot.
    If the quantum of the current DMA buffer is expired
    Submit next DMA buffer to the coprocessor.
    Reset the quantum as being running (not expired).
    Choose the next DMA buffer to execute.
        If the DMA buffer requires paging, submit it to the paging thread.
        Else, all memory resources are already present, just insert the DMA buffer in the ready slot;
Else if the memory manager failed because the paging buffer is full
            Wait until the current DMA buffer's quantum end or finishes.
            Submit the paging buffer to the coprocessor.
            Wait until the paging buffer is done.
            Go back asking the memory manager to paged-in the remaining of the resource list.
            Else if the memory manager failed because there isn't enough available resource
If we've passed the last eviction policy
Undo the resource move, or run the paging buffer.
    Reject the DMA buffer.
    We're done.
    Else if the current eviction policy is above application interference.
    If the DMA buffer hasn't been split yet.
    Split the DMA buffer at the closest point to the current paged-in resources.
    If no more resources are needed
    Move the paging buffer and split DMA buffer to the ready-to-execute slot.
    Move the remaining DMA buffer back to the head of the ready queue for the context.
If the quantum of the current DMA buffer is expired,
Submit next DMA buffer to the coprocessor.
Reset the quantum as being running (not expired).
    Choose the next DMA buffer to execute.
    If the DMA buffer requires paging, submit it to the paging thread.
        Else, all memory resources are already present; just insert the DMA buffer in the ready slot;.
    Ask VidMm to mark candidate for eviction using the current policy.
    If VidMm returns an error saying no memory could be marked with the current policy,
        Increase the eviction policy.
        Go back to the start of the eviction policy check.
            Else, some memory was marked.
            Go back to trying to page in the resources.
```

FIGURE 12(B)

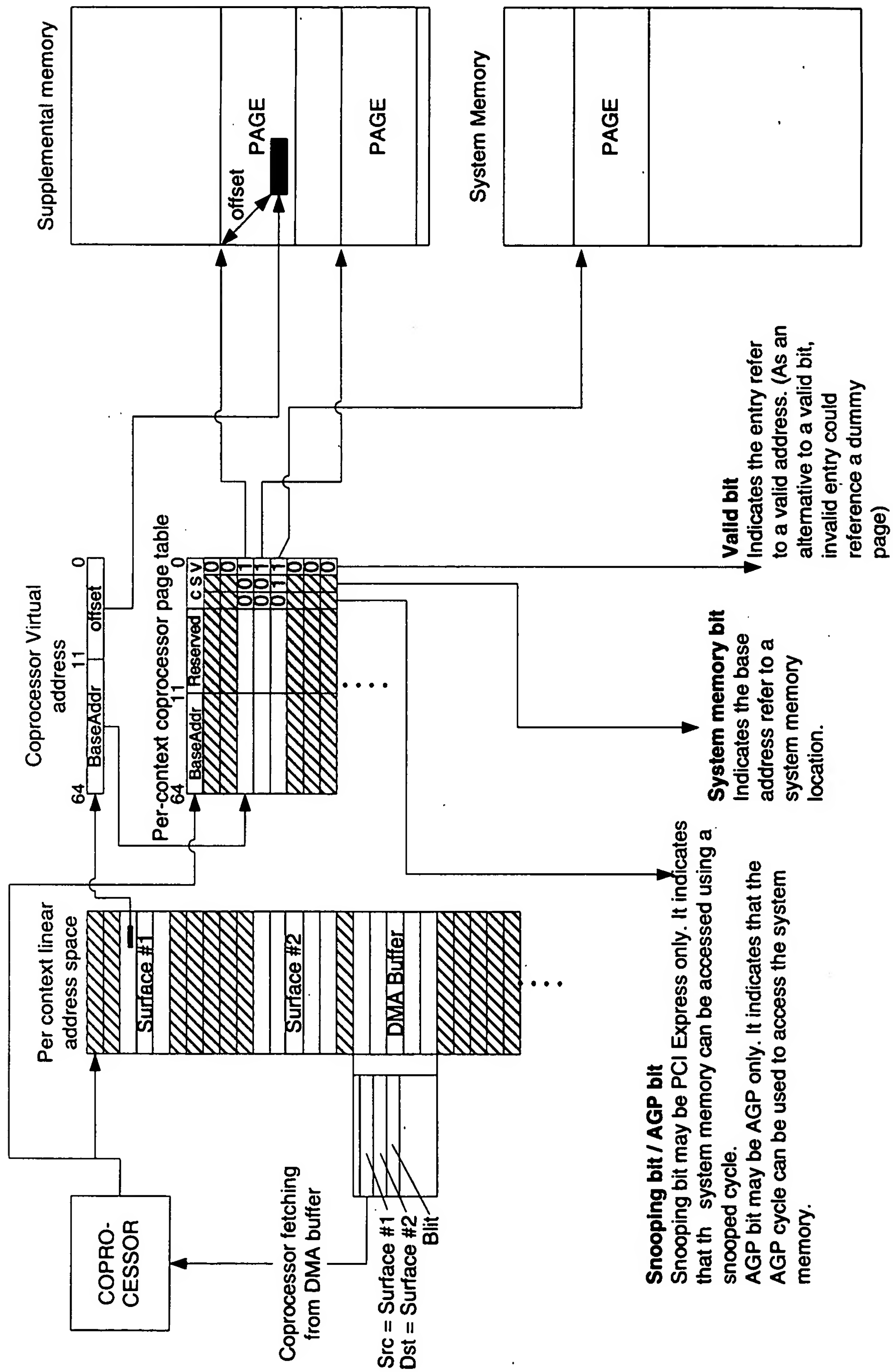

Supplemental memory
PAGE
offset
PAGE
System Memory
PAGE
Coprocessor Virtual address
64
BaseAddr
11
offset
0
Per-context coprocessor page table
64
BaseAddr
11
Reserved
0
c s v
0 0 0
0 0 1
0 0 1
0 1 1
0 0 0
0 0 0
Valid bit
Indicates the entry refer to a valid address. (As an alternative to a valid bit, invalid entry could reference a dummy page)
System memory bit
Indicates the base address refer to a system memory location.
Per context linear address space
Surface #1
Surface #2
DMA Buffer
COPRO-CESSOR
Coprocessor fetching from DMA buffer
Src = Surface #1
Dst = Surface #2
Blit
Snooping bit / AGP bit
Snooping bit may be PCI Express only. It indicates that th system memory can be accessed using a snooped cycle.
AGP bit may be AGP only. It indicates that the AGP cycle can be used to access the system memory.

FIGURE 13

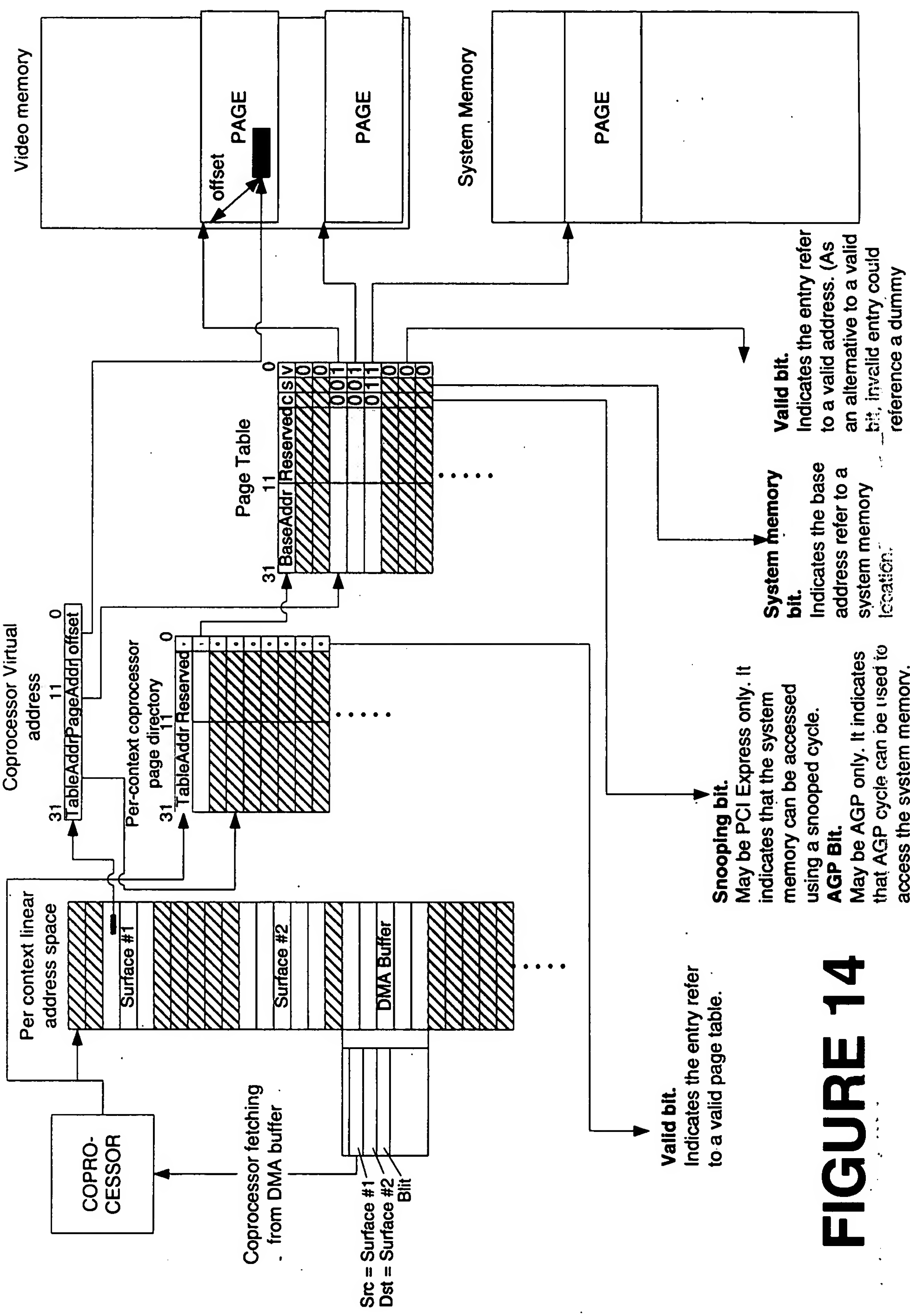

Video memory
offset
PAGE
PAGE
System Memory
PAGE
Page Table
31
11
0
BaseAddr
Reserved
c
s
v
Coprocessor Virtual address
TableAddr
PageAddr
offset
Per-context coprocessor page directory
TableAddr
Reserved
Per context linear address space
Surface #1
Surface #2
DMA Buffer
COPRO-CESSOR
Coprocessor fetching from DMA buffer
Src = Surface #1
Dst = Surface #2
Blit
Valid bit.
Indicates the entry refer to a valid page table.
Snooping bit.
May be PCI Express only. It indicates that the system memory can be accessed using a snooped cycle.
AGP Bit.
May be AGP only. It indicates that AGP cycle can be used to access the system memory.
System memory bit.
Indicates the base address refer to a system memory location.
Valid bit.
Indicates the entry refer to a valid address. (As an alternative to a valid bit, invalid entry could reference a dummy page)

FIGURE 14

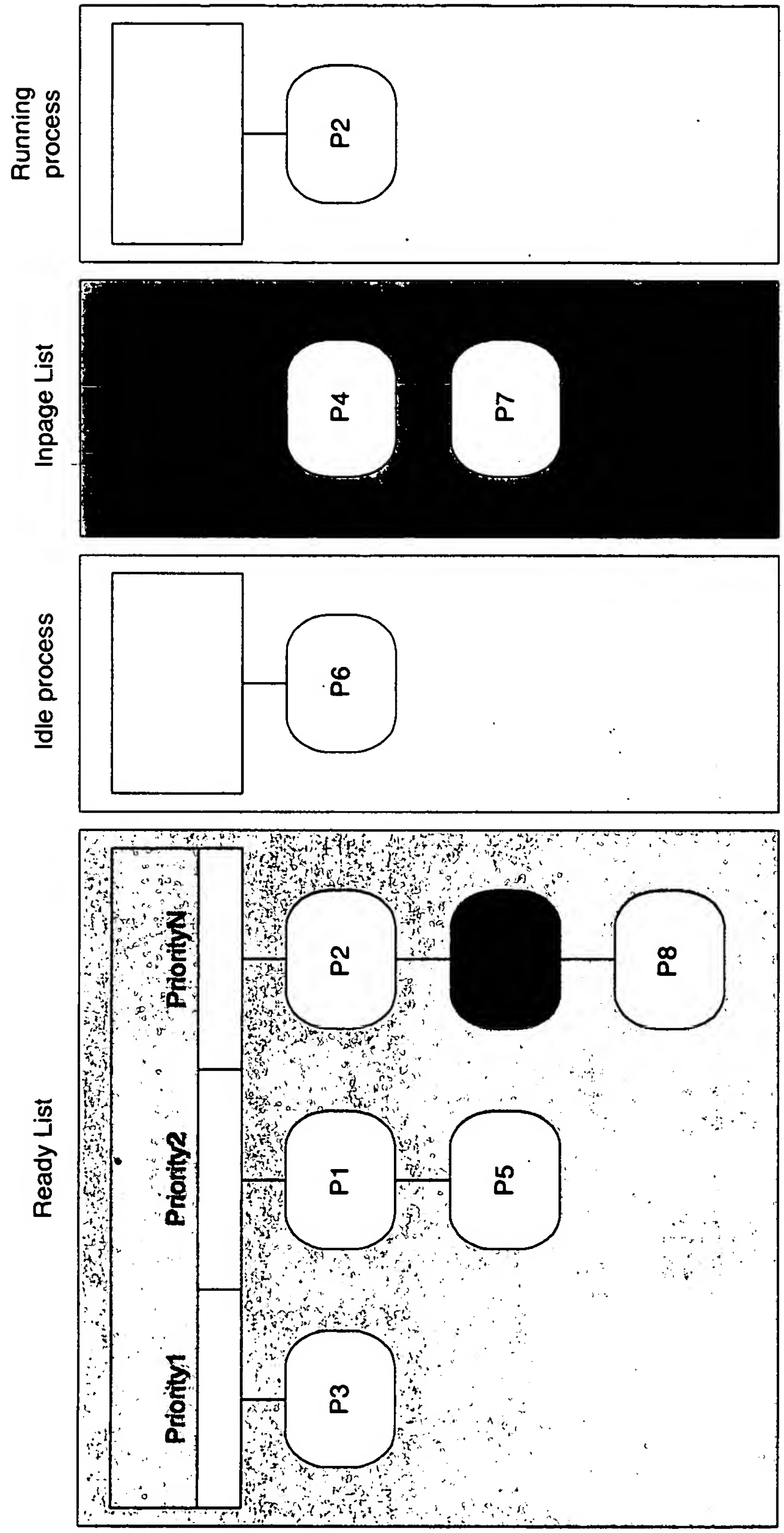
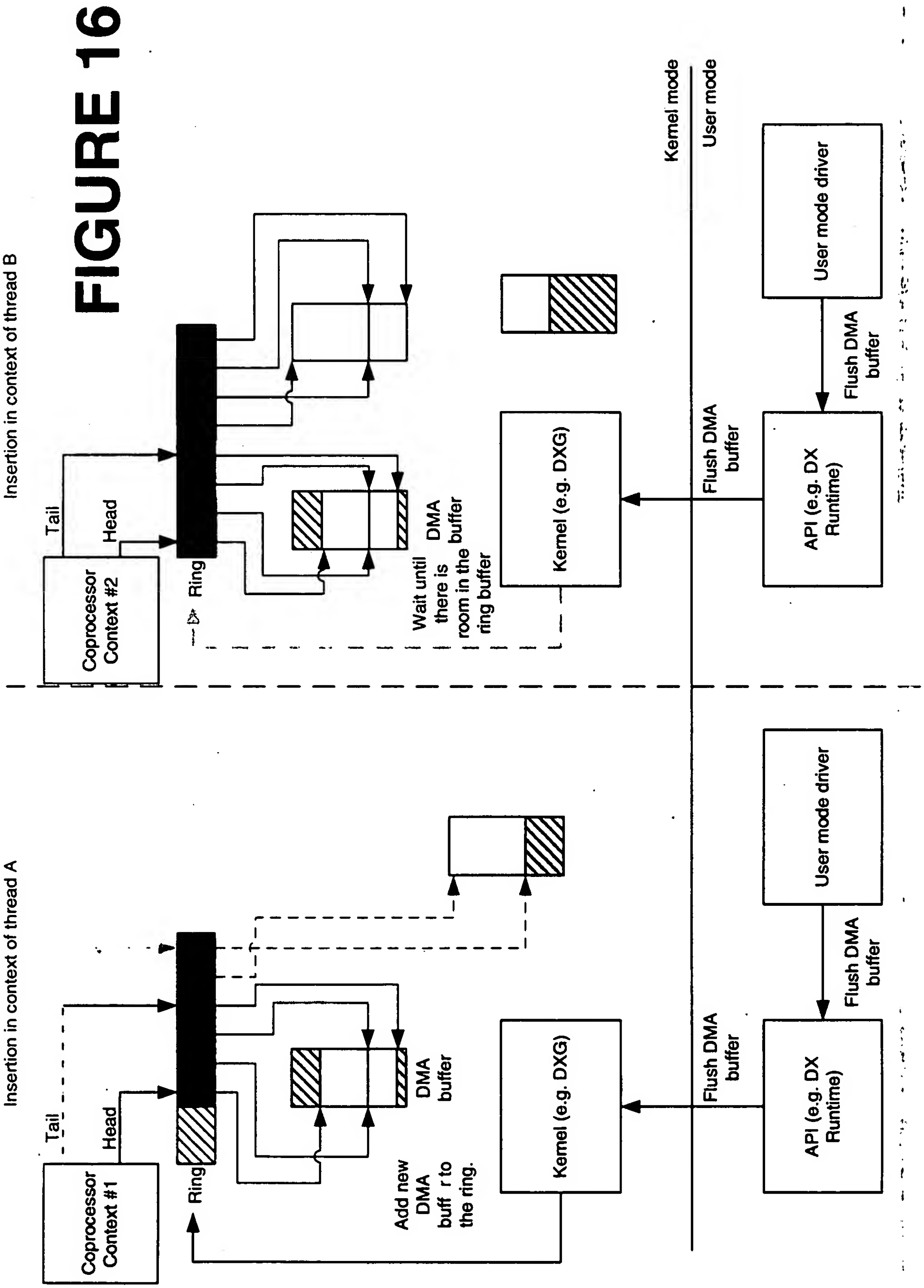

Running process
P2
Inpage List
P4
P7
Idle process
P6
Ready List
PriorityN
Priority2
Priority1
P2
P8
P1
P5
P3

FIGURE 15

FIGURE 16


Insertion in context of thread A
Tail
Head
Coprocessor Context #1
Ring
DMA buffer
Add new DMA buff r to the ring.
Kernel (e.g. DXG)
Flush DMA buffer
API (e.g. DX Runtime)
Flush DMA buffer
User mode driver
Insertion in context of thread B
Tail
Head
Coprocessor Context #2
Ring
DMA buffer
Wait until there is room in the ring buffer
Kernel (e.g. DXG)
Kernel mode
User mode
Flush DMA buffer
API (e.g. DX Runtime)
Flush DMA buffer
User mode driver

PROCESS A: Submit (IRQL passive, rendering thread context, coprocessor Context mutex held)

```
Acquire the VIDMM_lock.
Process the list of resources given, and update the usage information about allocations in this process.
Release the VIDMM lock.
Take the scheduler lock.
Call the driver to insert the current DMA buffer into the ring.
If the driver succeeded.
    If the context was idle.
            Insert the context back in the ready list at the tail of the queue for its current priority.
    If there is no context transfer pending and the current context is lower priority than the current context.
            Call the driver to context switch to this context.
            Signal that a context switch is pending.
Release the scheduler lock.
If the driver failed, the ring was full.
    Wait on an event that will be signaled when room becomes available.
    After the wait, go back to acquiring the scheduler lock.
If there is enough room left in the DMA buffer for another submission.
    Return to user mode with the current DMA buffer.
Acquire a new DMA buffer from the context's pool.
If DMA pool couldn't give another buffer at this time
    Wait on an event that will be signaled when a DMA buffer is inserted back into the pool.
    When the wait is over, go back to trying to get a new DMA buffer.
Return the new DMA buffer to user mode.
```

PROCESS B: Context switch done (IRQL device, any thread context)

```
Take the scheduler lock.
    If a higher priority context is now ready for execution.
            Call the driver to context switch to the highest priority context.
    Else
            Signal that no context switch is currently pending.
Release the scheduler lock
```

PROCESS C: Quantum expires (IRQL device, any thread context)

```
Take the scheduler lock.
Reset the current priority of the context to its base priority.
Insert the context back at the end of the queue for its current priority.
If no context switches are currently pending.
    Ask the driver to do a context switch to the highest priority context.
Release the scheduler lock.
```

FIGURE 17(A)

PROCESS D: Task finishes (IRQL device, any thread context)

Take the scheduler lock.
Ask the driver whether the context is really empty.
If the context is really empty.
  Reset the current priority of the context to its base priority.
  Insert the context in the idle list.
If the context wasn't really empty.
  If no context switches are currently pending.
    Ask the driver to do a context switch to the highest priority context.
Release the scheduler lock.

PROCESS E: Page Fault (IRQL device, any thread context)

Take the scheduler lock.
Remove the context from the ready list.
Insert the context in the in page list as an atomic operation.
If in page thread currently sleeping.
  Queue a DPC to signal to wakeup the worker thread.
If no context switch are currently pending.
  Ask the driver to do a context switch to the highest priority context.
Release the scheduler lock.

PROCESS F: Fault resolved (IRQL device, any thread context)

Take the scheduler lock.
Remove the context from the in page list.
Insert the context back in the ready list for its current priority.
If co context switches are currently pending, and the current context is higher priority than the currently running context.
  Ask the driver to do a context switch to the highest priority context.
Release the scheduler lock.

FIGURE 17(B)

PROCESS G: In page worker thread

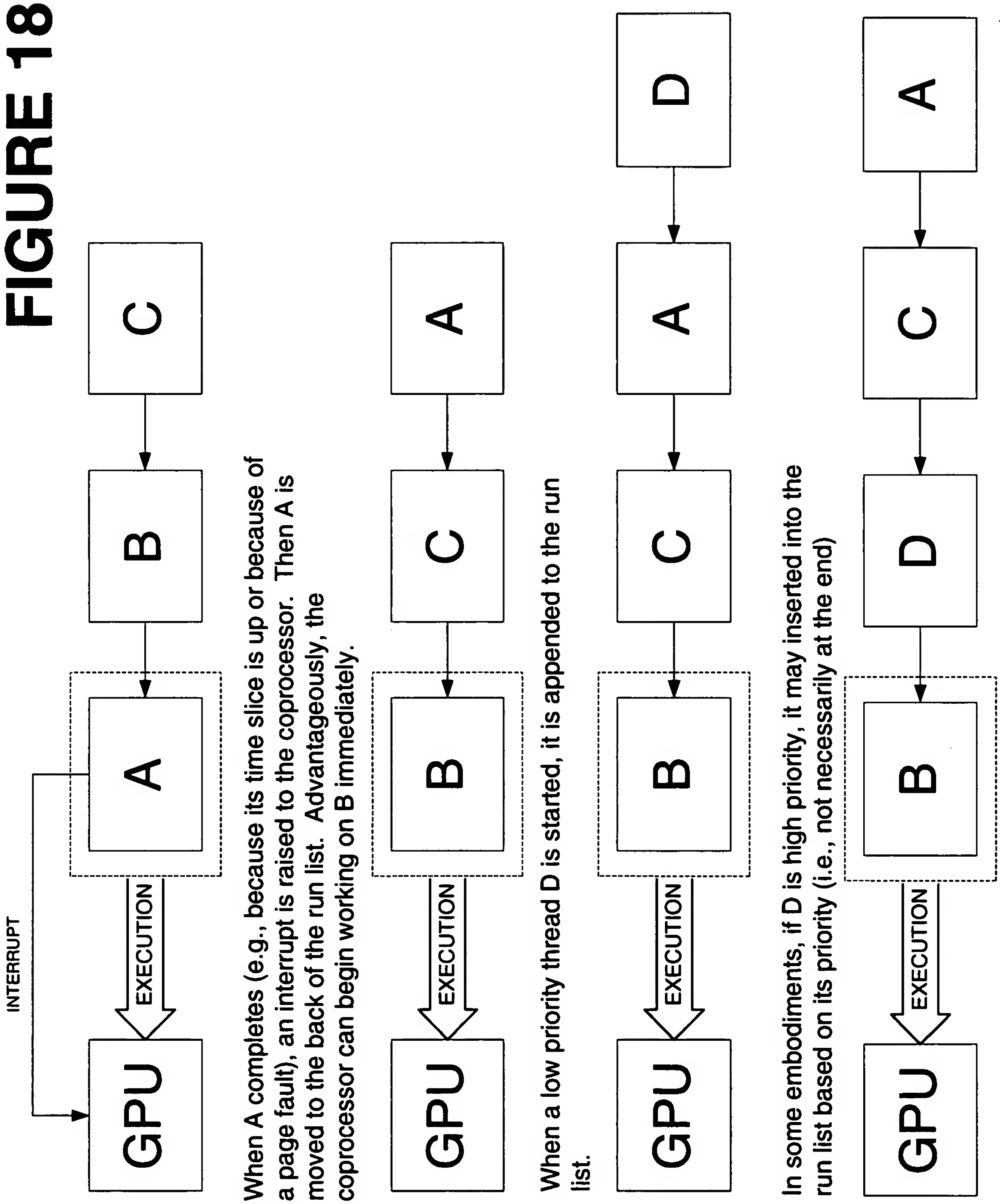
Go through the list of contexts in the inpage queue. Pick up the highest priority one.
Ask the driver for the list of resources required to make forward progress on the context.
Take the VIDMM lock.
Find a location for each of the allocations required for forward progress.
Invalidate the virtual address or handle for the allocation getting evicted.
Ask the driver to fill a DMA buffer with the memory transfer commands necessary to bring the required allocations to their selected spots.
Release the VIDMM lock.
Submit the VidMm context as a regular coprocessor context.
If the list of contexts is empty, sleep until an item gets added.
Go back to the beginning of the loop.

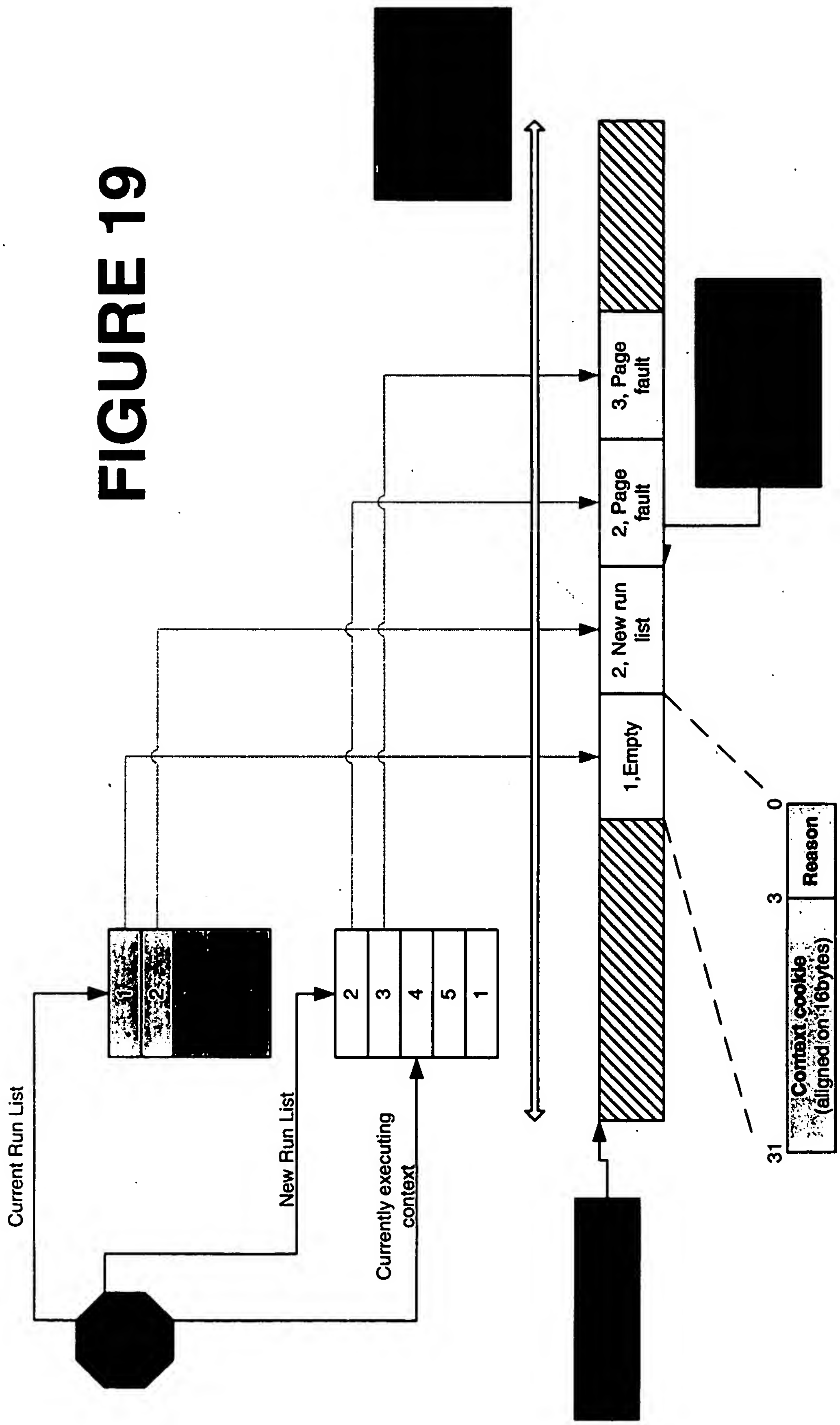
PROCESS H: Periodic timer (passive level, system thread context)

Take the scheduler lock.
Increase the current priority of each context.
Release the scheduler lock.

FIGURE 17(C)

FIGURE 18


INTERRUPT
C
B
A
EXECUTION
GPU
When A completes (e.g., because its time slice is up or because of a page fault), an interrupt is raised to the coprocessor. Then A is moved to the back of the run list. Advantageously, the coprocessor can begin working on B immediately.
A
C
B
EXECUTION
GPU
When a low priority thread D is started, it is appended to the run list.
D
A
C
B
EXECUTION
GPU
In some embodiments, if D is high priority, it may inserted into the run list based on its priority (i.e., not necessarily at the end)
A
C
D
B
EXECUTION
GPU



[31:4] Context cookie: Various embodiments may use a 32 bit value, aligned on 16 bytes, associated with a context (part of a context definition). This allows a scheduler to identify the context quickly.

[3:0] Reason: Reason for the context switch:

0: Page fault
1: Empty
2: Synchronization object
3: New run list
4: General protection fault.

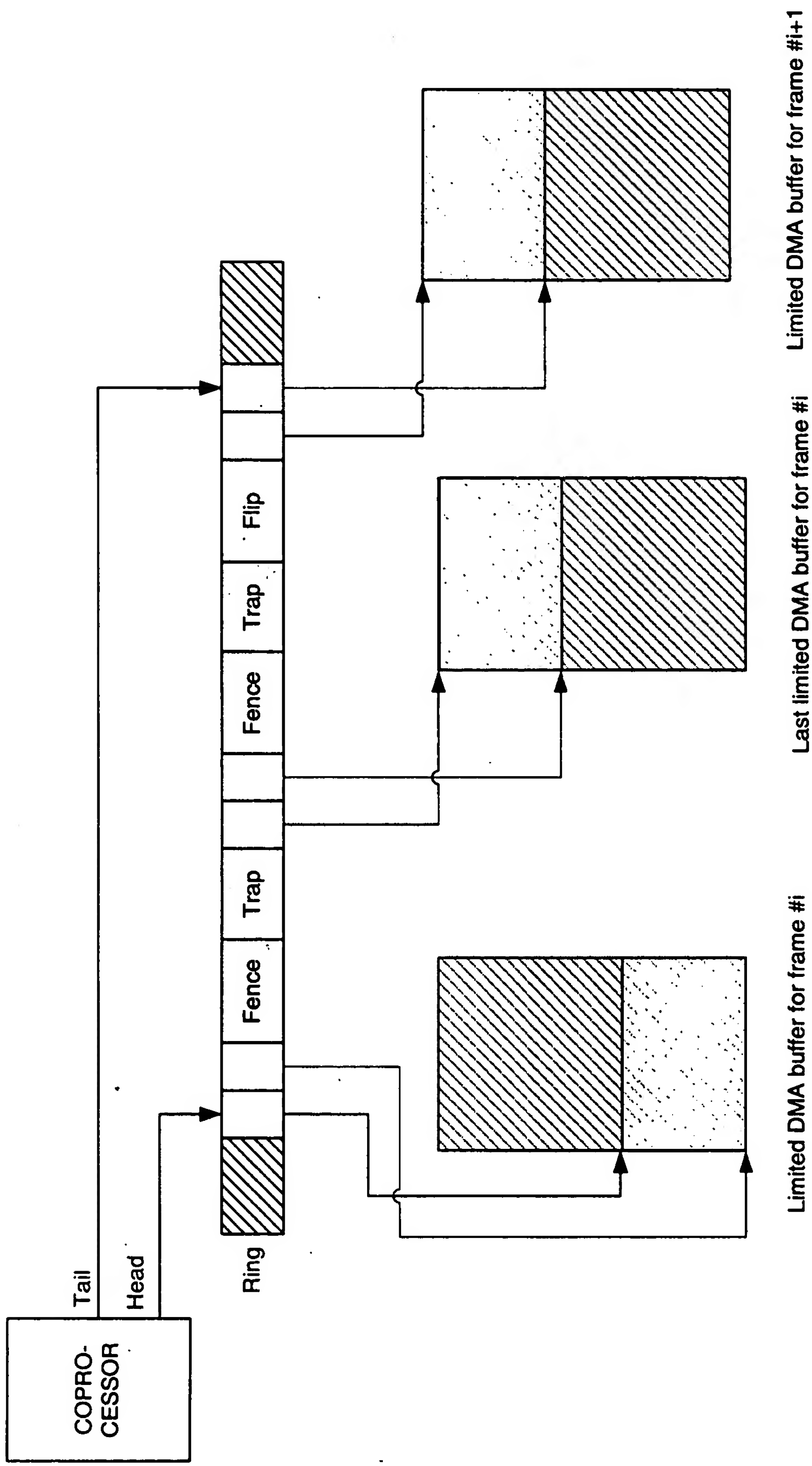

COPRO-
CESSOR
Tail
Head
Ring
Fence
Trap
Fence
Trap
Flip
Limited DMA buffer for frame #i
Last limited DMA buffer for frame #i
Limited DMA buffer for frame #i+1

FIGURE 20

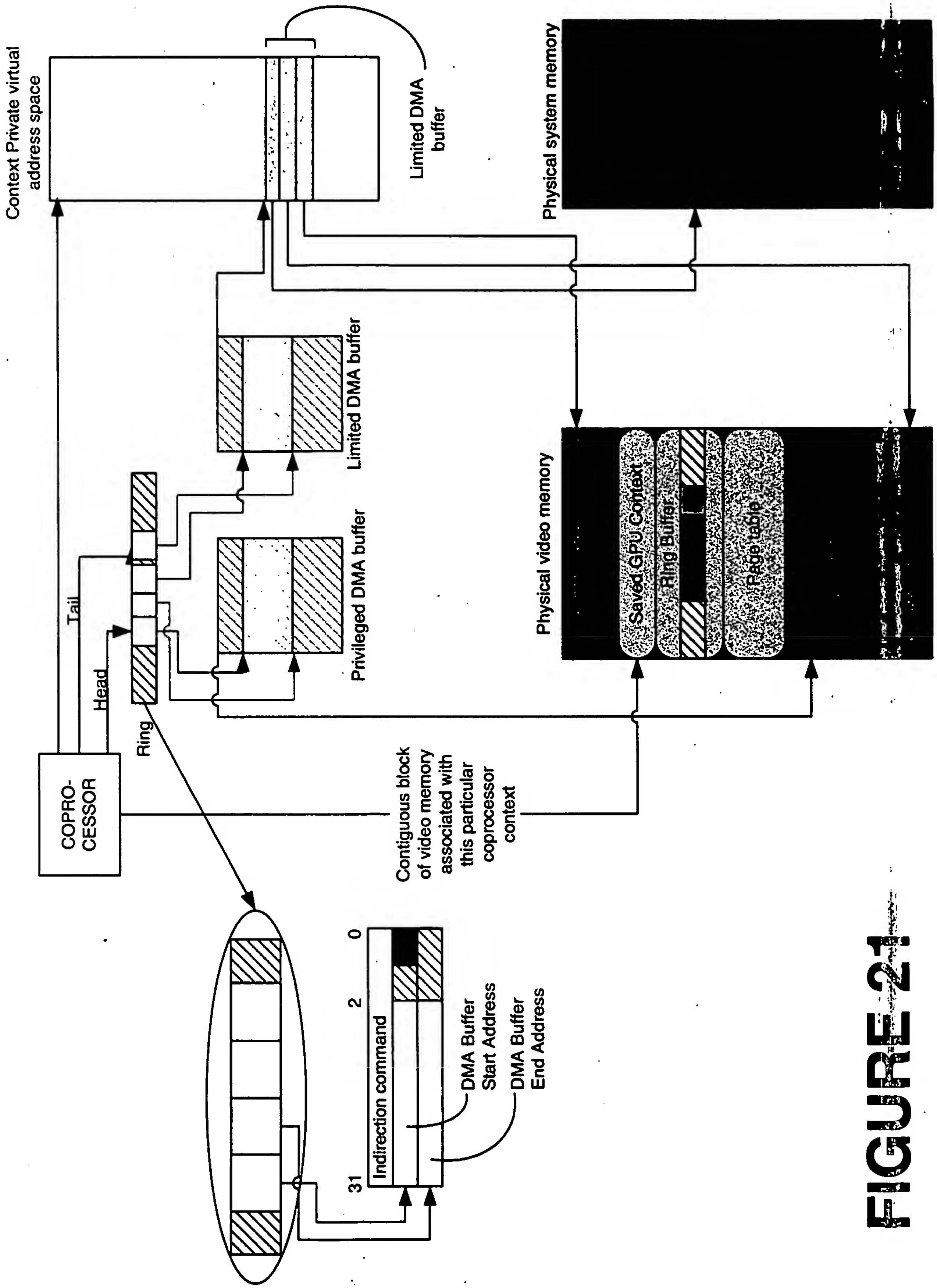

Context Private virtual address space
Limited DMA buffer
Physical system memory
Limited DMA buffer
Privileged DMA buffer
Physical video memory
Saved GPU Context
Ring Buffer
Page table
Tail
Head
Ring
COPRO-CESSOR
Contiguous block of video memory associated with this particular coprocessor context
0
2
31
Indirection command
DMA Buffer Start Address
DMA Buffer End Address

FIGURE 21

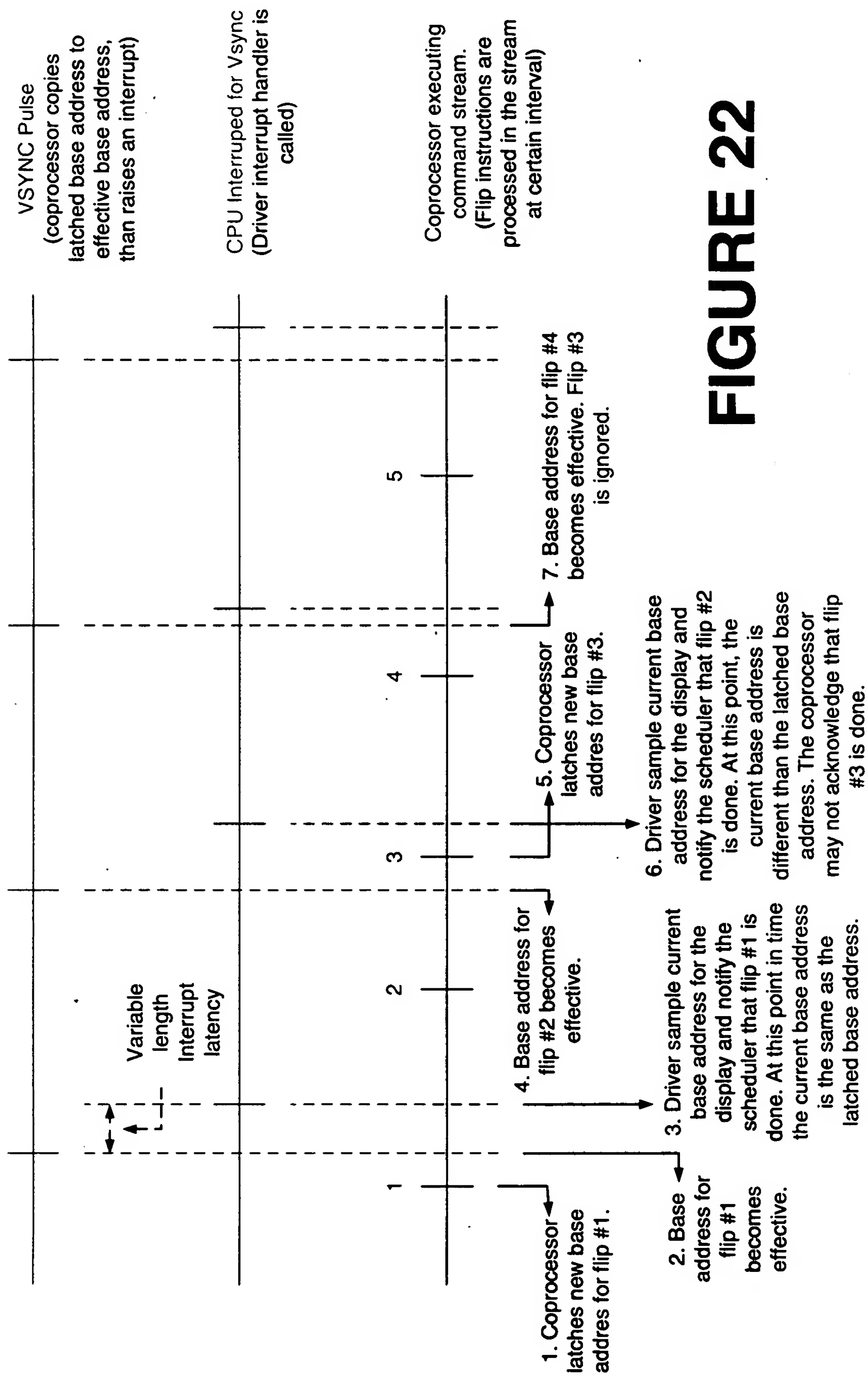

VSYNC Pulse (coprocessor copies latched base address to effective base address, than raises an interrupt)
CPU Interruped for Vsync (Driver interrupt handler is called)
Coprocessor executing command stream. (Flip instructions are processed in the stream at certain interval)
Variable length Interrupt latency
1
2
3
4
5
1. Coprocessor latches new base addres for flip #1.
2. Base address for flip #1 becomes effective.
3. Driver sample current base address for the display and notify the scheduler that flip #1 is done. At this point in time the current base address is the same as the latched base address.
4. Base address for flip #2 becomes effective.
5. Coprocessor latches new base addres for flip #3.
6. Driver sample current base address for the display and notify the scheduler that flip #2 is done. At this point, the current base address is different than the latched base address. The coprocessor may not acknowledge that flip #3 is done.
7. Base address for flip #4 becomes effective. Flip #3 is ignored.

FIGURE 22

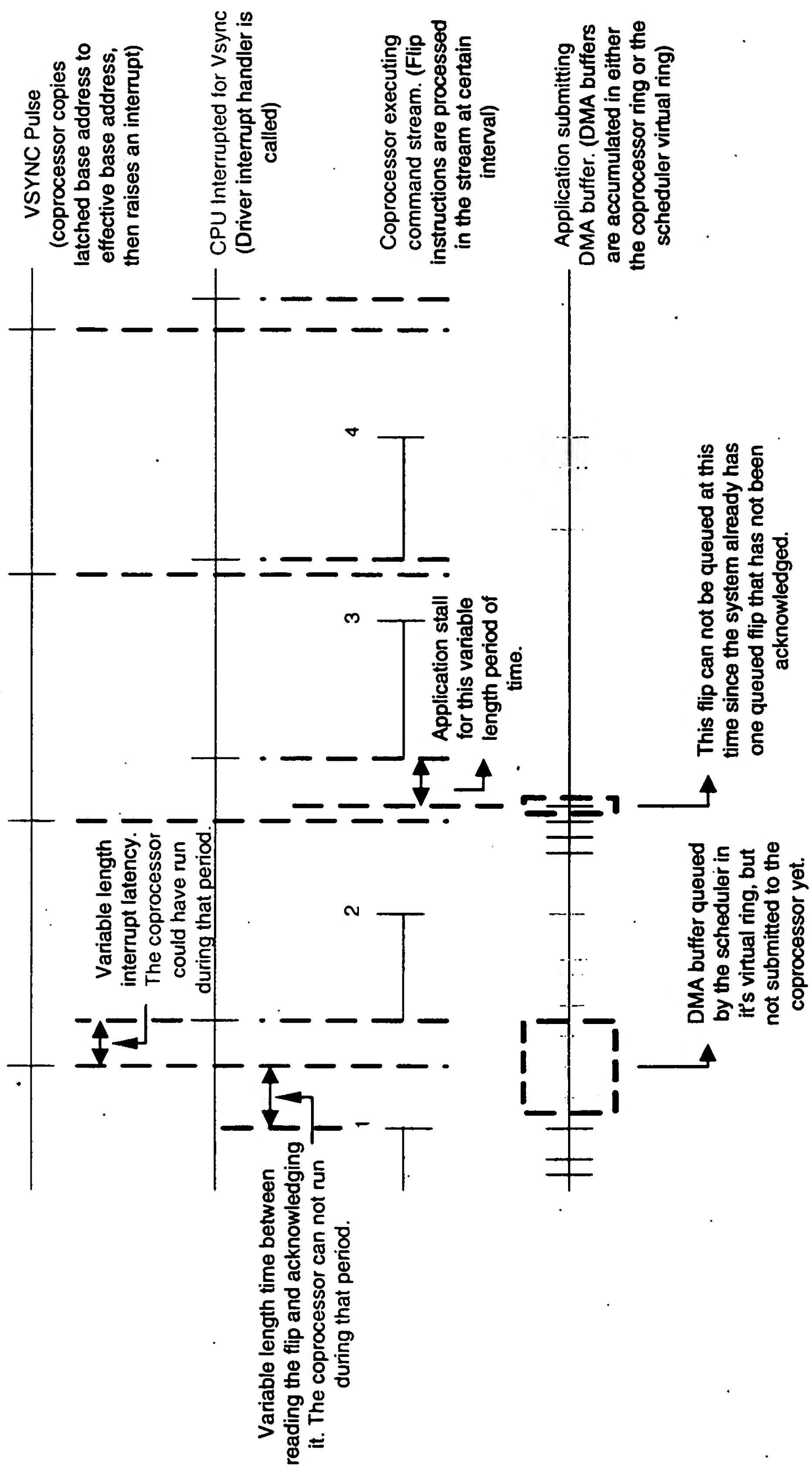

VSYNC Pulse (coprocessor copies latched base address to effective base address, then raises an interrupt)
CPU Interrupted for Vsync (Driver interrupt handler is called)
Coprocessor executing command stream. (Flip instructions are processed in the stream at certain interval)
Application submitting DMA buffer. (DMA buffers are accumulated in either the coprocessor ring or the scheduler virtual ring)
4
3
2
Application stall for this variable length period of time.
This flip can not be queued at this time since the system already has one queued flip that has not been acknowledged.
Variable length interrupt latency. The coprocessor could have run during that period.
DMA buffer queued by the scheduler in it's virtual ring, but not submitted to the coprocessor yet.
Variable length time between reading the flip and acknowledging it. The coprocessor can not run during that period.

FIGURE 23

Coprocessor Thread A

Pseudo code:

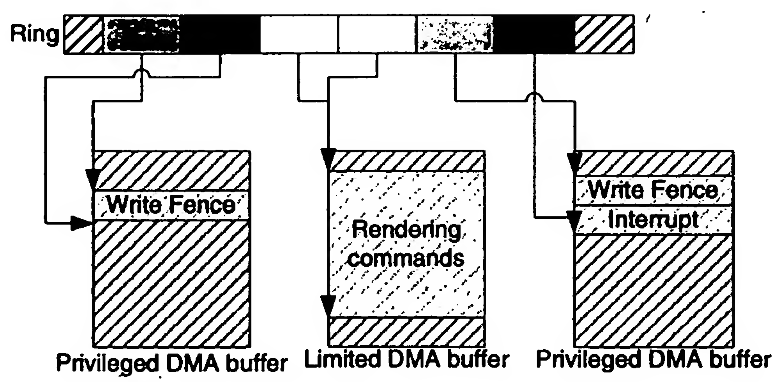
```
// Wait until we have exclusive access to the shared surface.
//
DxAcquireMutex(gSharedMutex);

// Set the shared surface as the render target.
//
DxSetRenderTarget(gSharedSurface);

// Render what we need in the shared surface.
//
DxDrawSomething();

// We're done with rendering, release the mutex.
//
DxReleaseMutex(gSharedMutex)
```

Coprocessor stream:



Coprocessor Thread B

Pseudo code:

```
// Wait until we have exclusive access to the shared surface.
//
DxAcquireMutex(gSharedMutex);

// Set the shared surface as a texture.
//
DxSetTexture(gSharedSurface);

// Render what we need with the shared surface.
//
DxDrawSomething();

// We're done with rendering, release the mutex.
//
DxReleaseMutex(gSharedMutex)
```

Coprocessor stream:

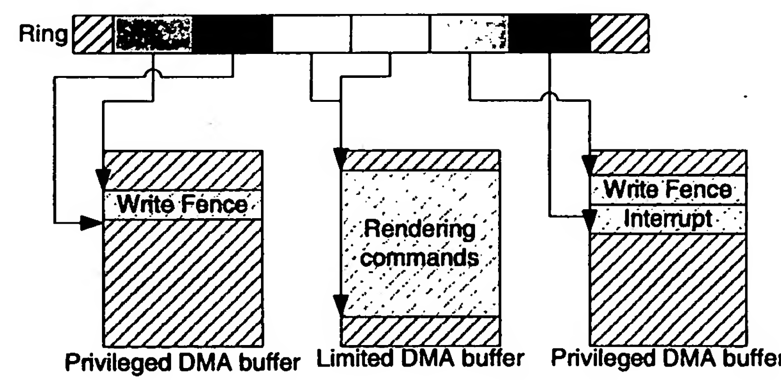


FIGURE 24

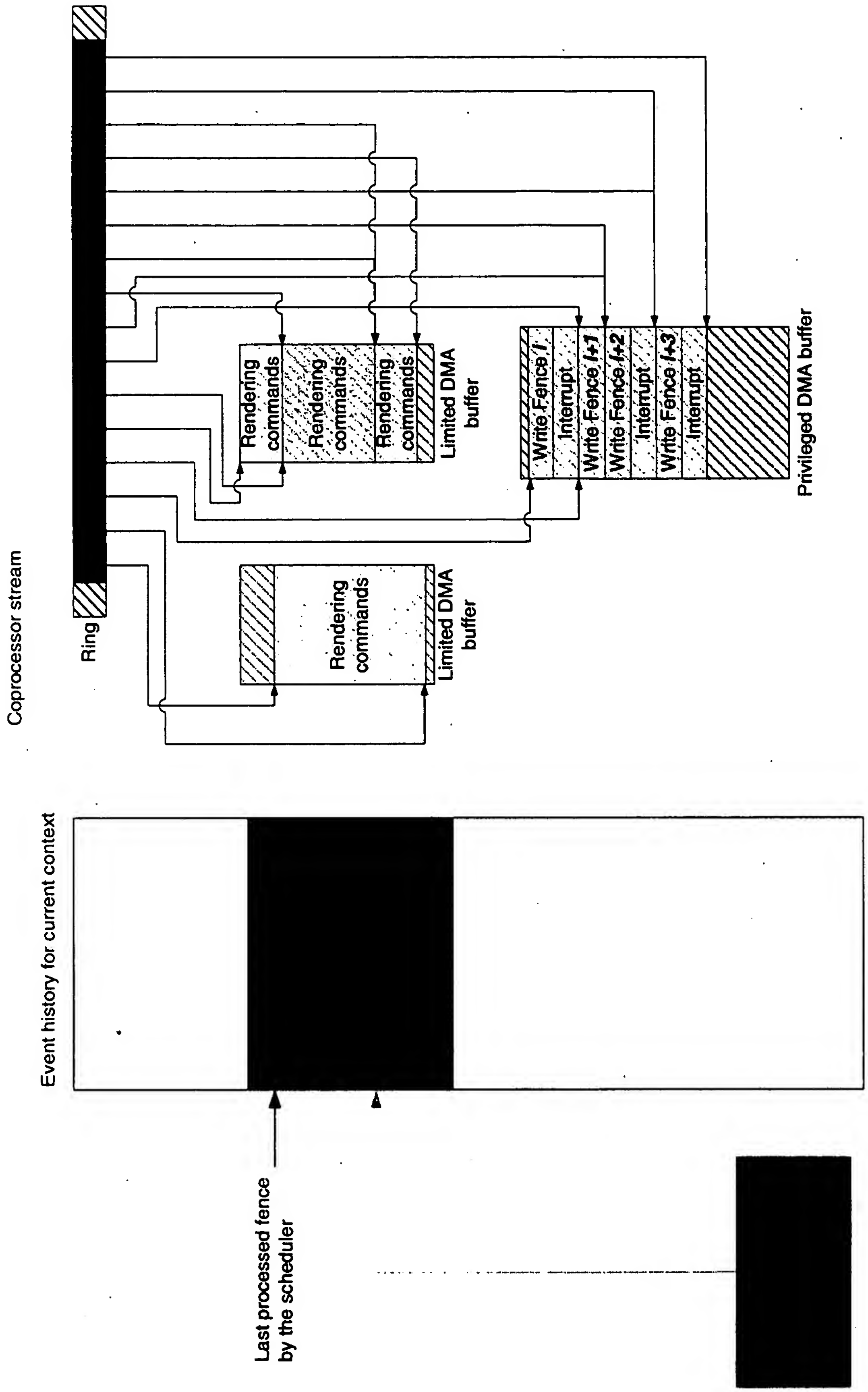

Coprocessor stream
Ring
Rendering commands
Limited DMA buffer
Rendering commands
Rendering commands
Rendering commands
Limited DMA buffer
Write Fence I
Interrupt
Write Fence I+1
Write Fence I+2
Interrupt
Write Fence I+3
Interrupt
Privileged DMA buffer
Event history for current context
Last processed fence by the scheduler

FIGURE 25

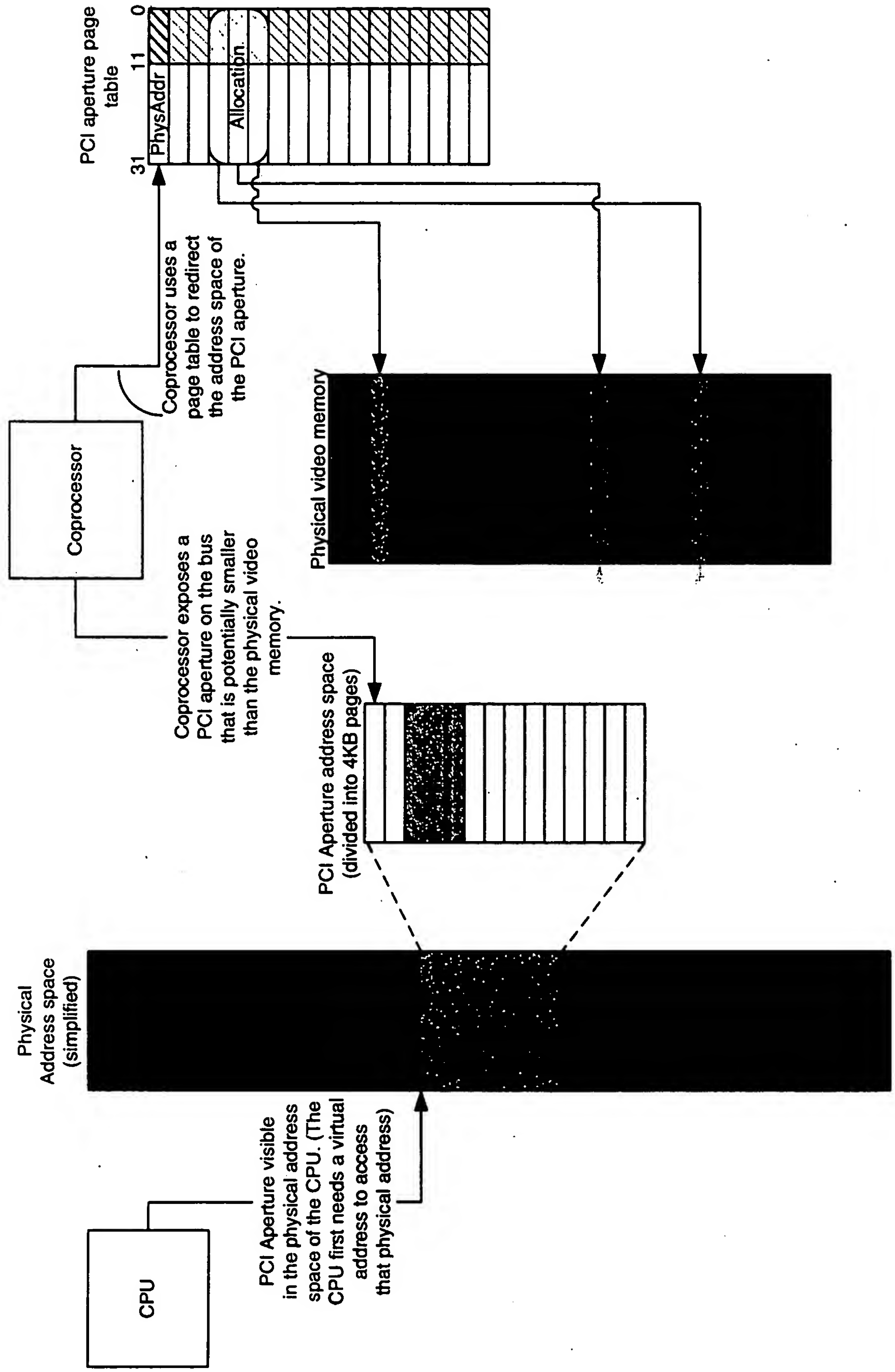

PCI aperture page table
0
11
31
PhysAddr
Allocation
Coprocessor uses a page table to redirect the address space of the PCI aperture.
Coprocessor
Physical video memory
Coprocessor exposes a PCI aperture on the bus that is potentially smaller than the physical video memory.
PCI Aperture address space (divided into 4KB pages)
Physical Address space (simplified)
PCI Aperture visible in the physical address space of the CPU. (The CPU first needs a virtual address to access that physical address)
CPU

FIGURE 26